SUST PROJECT fumehood v2

Backend Components:

```
Api.py Documentation
==========================
Overview
------------
The Api.py file contains a class Api that provides a simple interface for making API calls. I
t is designed to interact with a specific API endpoint, retrieving data based on user-specifi
ed parameters.
Class Attributes
-------------------
url: The base URL of the API endpoint.
headers: A dictionary of headers to be sent with the API request.
Methods
------------
__init__(config_file='config.json')
Initializes the Api class with a configuration file.
Loads the configuration file and extracts the API endpoint URL and headers.
Sets the url and headers attributes.
call(TLInstance, start_day=None, end_day=None, size=1)
Makes an API call to the specified endpoint.
Parameters:
TLInstance: The TLInstance parameter.
start_day: The start date for the query (optional).
end_day: The end date for the query (optional).
size: The size parameter for the query (default: 1).
Returns a list of hits from the API response.
Example Usage
-----------------
Notes
----------
The config.json file is expected to contain the API endpoint URL and headers.
The call method returns a list of hits from the API response.
The main function demonstrates an example API call.
Error handling is implemented to catch and print any exceptions that occur during the API cal
l.
```

```
**Cleaner.py Documentation**
==========================

**Overview**
------------

The `Cleaner.py` script is designed to clean and preprocess data from a CSV file. It performs
various operations such as data validation, transformation, and merging with an external lis
t.

**Functions**
-------------

### `validate(df)`

*   **Purpose:** Validate data by checking if the length of the `sort` column is greater than
or equal to a calculated value `n`.
```

```
*   **Input:** A pandas DataFrame `df`.
*   **Output:** The input DataFrame with a new column `valid` indicating whether the data is valid (1) or not (0).

### `transform(df)`

*   **Purpose:** Transform data by scaling the `data` column based on the `percent` column.
*   **Input:** A pandas DataFrame `df`.
*   **Output:** The input DataFrame with the `data` column transformed.

### `date_transform(df)`

*   **Purpose:** Convert the `time` column to datetime format and extract the date.
*   **Input:** A pandas DataFrame `df`.
*   **Output:** The input DataFrame with a new column `date` in datetime format.

### `period(df)`

*   **Purpose:** Create a new column `overnight` indicating whether the time is between 00:00:00 and 08:00:00.
*   **Input:** A pandas DataFrame `df`.
*   **Output:** The input DataFrame with a new column `overnight`.

### `clean(df)`

*   **Purpose:** Perform a series of cleaning operations, including data validation, transformation, and merging with an external list.
*   **Input:** A pandas DataFrame `df`.
*   **Output:** A tuple containing the cleaned DataFrame and a DataFrame with invalid data.

**Usage**
-----

To use the `Cleaner.py` script, simply import it and call the `clean()` function, passing in a pandas DataFrame as an argument.

```python
import pandas as pd
from Cleaner import clean

# Load data from CSV file
df = pd.read_csv('data.csv')

# Clean data
clean_df, invalid_df = clean(df)

# Print cleaned data
print(clean_df)
```

Notes
------

*   The script assumes that the input CSV file has columns `fumehood`, `sort`, `percent`, `data`, `time`, and `TLInstance`.
*   The script uses the `pd.merge()` function to merge the input DataFrame with an external list `df_list`.
```

```
*   The script uses the `pd.to_datetime()` function to convert date and time columns to datet
ime format.
```

```
**DataCaller.py Documentation**
==========================

**Overview**
------------

The `DataCaller.py` script is designed to retrieve and process data from an API. It provides
functions to fetch raw data, format the data, and retrieve cleaned and latest data.

**Functions**
-------------

### `getRawDataByInstance(instance, start, end, size)`

*   **Purpose:** Fetch raw data from the API for a given instance.
*   **Input:**
    *   `instance`: The instance ID to fetch data for.
    *   `start`: The start date (optional).
    *   `end`: The end date (optional).
    *   `size`: The number of data points to fetch (default: 10000).
*   **Output:** A pandas DataFrame containing the raw data.

### `format(df)`

*   **Purpose:** Format the data by converting data types and performing other operations.
*   **Input:** A pandas DataFrame `df`.
*   **Output:** The formatted DataFrame.

### `getCleanedData(instance, start, end, size=10000)`

*   **Purpose:** Fetch cleaned data for a given instance.
*   **Input:**
    *   `instance`: The instance ID to fetch data for.
    *   `start`: The start date (optional).
    *   `end`: The end date (optional).
    *   `size`: The number of data points to fetch (default: 10000).
*   **Output:** A pandas DataFrame containing the cleaned data.

### `latest(instance, size=10)`

*   **Purpose:** Fetch the latest data for a given instance.
*   **Input:**
    *   `instance`: The instance ID to fetch data for.
    *   `size`: The number of latest data points to fetch (default: 10).
*   **Output:** A pandas DataFrame containing the latest data.

### `main()`

*   **Purpose:** The main entry point of the script.
*   **Input:** None.
*   **Output:** None.

**Usage**
-----
```

To use the `DataCaller.py` script, simply import it and call the desired functions, passing in the required arguments.

```python
import pandas as pd
from DataCaller import DataCaller

# Create an instance of the DataCaller class
data_caller = DataCaller()

# Fetch raw data for a given instance
raw_data = data_caller.get_raw_data_by_instance('678', None, None, 5)
print("Raw Data:")
print(raw_data)

# Fetch cleaned data for a given instance
cleaned_data = data_caller.get_cleaned_data('678', None, None)
print("Cleaned Data:")
print(cleaned_data)

# Fetch the latest data for a given instance
latest_data = data_caller.latest('678')
print("Latest Data:")
print(latest_data)
```

Notes
---

*   The script assumes that the API instance ID is '678' and the start and end dates are None.
*   The script uses the `pd.to_datetime()` function to convert date and time columns to datetime format.
*   The script uses the `pd.Timedelta()` function to add a time offset to the datetime values.

Model.py Documentation
==========

Overview

The `Model.py` script contains various functions for data processing and analysis. It includes functions for slicing off-work hours, calculating the top 10 OSH values, counting occurrences of data exceeding a threshold, calculating the longest non-use period, and classifying rows based on activity and OSH values.

Functions

`slice_offwork(df)`

*   **Purpose:** Slice off the off-work hours from the input DataFrame `df`.
*   **Input:** A pandas DataFrame `df`.
*   **Output:** A DataFrame containing only the off-work hours data.

```
### `Top10OSH(df)`

*   **Purpose:** Calculate the top 10 OSH values for each TLInstance and mark corresponding r
ows as 1.
*   **Input:** A pandas DataFrame `df`.
*   **Output:** The input DataFrame with additional columns 'Top10OSH' and 'OSH'.

### `CDO(df, threshold=10)`

*   **Purpose:** Count occurrences where 'data' > 550 for each fumehood and mark fumehoods wi
th count exceeding the specified threshold.
*   **Input:**
    *   `df`: A pandas DataFrame.
    *   `threshold`: The threshold value (default: 10).
*   **Output:** The input DataFrame with additional columns 'CDOValue' and 'CDOET'.

### `calculate_longest_nonuse(data)`

*   **Purpose:** Calculate the longest non-use period for each day in the input data.
*   **Input:** A pandas Series `data`.
*   **Output:** An integer value indicating whether the longest non-use period exceeds 1 day.

### `classify_row(row)`

*   **Purpose:** Classify a row based on activity and OSH values.
*   **Input:** A pandas Series `row`.
*   **Output:** A string value indicating the classification.

### `classification(df)`

*   **Purpose:** Classify rows in the input DataFrame based on activity and OSH values.
*   **Input:** A pandas DataFrame `df`.
*   **Output:** The input DataFrame with an additional column 'cat'.
```

```
**VirtualBase.py Documentation**
==========================

**Overview**
------------

The `VirtualBase.py` script serves as the base for a virtual data processing system. It impor
ts necessary modules, reads a list of fumehood instances from a CSV file, and defines functio
ns for initializing and updating the data.

**Functions**
-------------

### `init()`

*   **Purpose:** Initialize the data processing by fetching cleaned data for each fumehood in
stance and concatenating the results.
*   **Input:** None
*   **Output:** A concatenated DataFrame `df` containing the cleaned data and a list `fail` o
f instances that failed to fetch data.

### `update()`
```

```
*   **Purpose:** Update the data by cleaning, processing, and saving the data to a CSV file.
*   **Input:** None
*   **Output:** The number of failed instances `fail_count` and the number of invalid fumehoo
ds `invalid_count`.

### `main()`

*   **Purpose:** Call the `update()` function to perform the data update process.
*   **Input:** None
*   **Output:** None

**Usage**
-----

To use the `VirtualBase.py` script, simply run it as a Python script. The `main()` function w
ill be called automatically.

```python
python VirtualBase.py
```

Notes
------

*   The script assumes that the `DataCaller` module is available and contains the `getCleaned
Data()` function.
*   The script assumes that the `Cleaner` module is available and contains the `clean()` func
tion.
*   The script assumes that the `Model` module is available and contains the `Top10OSH()`, `C
DO()`, and `classification()` functions.
*   The script reads the fumehood instance list from a CSV file named `fumehood list.csv`.
*   The script saves the processed data to a CSV file named `database/data.csv`.

Dependencies
----

*   `pandas` for data manipulation and analysis
*   `DataCaller` for fetching cleaned data
*   `Cleaner` for cleaning and processing data
*   `Model` for data analysis and classification
*   `datetime` for date and time manipulation
*   `os` for file system operations
```

```
**VirtualPlot.py Documentation**
==========================

**Overview**
------------

The `VirtualPlot.py` script is designed to create plots for fumehood data using Matplotlib. I
t defines a function `pplot()` to generate plots based on the input data and mode.

**Function**
------------

### `pplot(df, mode)`
```

```
*   **Purpose:** Generate a plot for the input data `df` based on the specified `mode`.
*   **Input:**
    + `df`: A pandas DataFrame containing the fumehood data.
    + `mode`: A string indicating the plot mode (either 'OSH', 'CDO', or 'Temp').
*   **Output:**
    + If `mode` is 'Temp', returns a BytesIO buffer containing the plot image.
    + Otherwise, saves the plot to a file in the specified directory.

**Plot Modes**
--------------

*   **OSH (Overnight Sash Height)**: Plots the data with a title indicating the overnight sas
h height.
*   **CDO (Above Safe Height Count)**: Plots the data with a title indicating the above safe
height count.
*   **Temp (Temporary Plot)**: Returns a BytesIO buffer containing the plot image.

**Plot Customization**
----------------------

*   The plot includes a line plot of the data, with hourly ticks and labels.
*   The plot includes vertical spans for working hours (8am-12am).
*   The plot includes horizontal lines at y-values 800, 500, and 100.
*   The plot includes a title with the fumehood name, room, and department.
*   The plot includes labels for the x-axis (time) and y-axis (data).
```

```
**app.py Documentation**
========================

**Overview**
------------

The `app.py` script is a Flask web application that provides various routes for interacting w
ith the fumehood data. It uses the `VirtualPlot` and `DataCaller` modules to generate plots a
nd retrieve data.

**Routes**
---------

### `/`

*   **Purpose:** Render the index template with a list of unique fumehood numbers.
*   **Method:** GET
*   **Template:** `index.html`

### `/get_image/<fumehood_number>`

*   **Purpose:** Generate a temporary plot for the specified fumehood number and return it as
an image.
*   **Method:** GET
*   **Parameters:** `fumehood_number` (string)
*   **Template:** None

### `/latest`

*   **Purpose:** Retrieve the latest time for each fumehood and render the latest template wi
th the results.
```

```
*   **Method:** GET
*   **Template:** `latest.html`

### `/catch`

*   **Purpose:** Render the OSH template with a list of fumehood numbers that have a Top10OSH
value of 1.
*   **Method:** POST
*   **Template:** `OSH.html`

### `/update`

*   **Purpose:** Update the database and return a success message.
*   **Method:** POST
*   **Template:** None

### `/search`

*   **Purpose:** Search for data within a specified date range and return a plot as an image.
*   **Method:** POST
*   **Parameters:** `fumehood` (string), `start_date` (string), `end_date` (string)
*   **Template:** None

### `/danger`

*   **Purpose:** Render the CDO template with a list of fumehood numbers that have a CDOET va
lue of 1.
*   **Method:** POST
*   **Template:** `CDO.html`

### `/cate`

*   **Purpose:** Render the category template with a list of fumehood numbers based on the se
lected category.
*   **Method:** POST
*   **Parameters:** `option` (string)
*   **Template:** `cat.html`

### `/data.csv`

*   **Purpose:** Download the data.csv file.
*   **Method:** GET
*   **Template:** None

### `/get-schema`

*   **Purpose:** Return the schema of the data.csv file.
*   **Method:** GET
*   **Template:** None

### `/historical`

*   **Purpose:** Download the historical.parquet file.
*   **Method:** GET
*   **Template:** None

**Functions**
------------
```

```
### `zip_lists(a, b)`

*   **Purpose:** Zip two lists together.
*   **Method:** Template filter
*   **Parameters:** `a` (list), `b` (list)
*   **Return:** Zipped list

**Variables**
------------

### `data`

*   **Purpose:** Global variable containing the formatted data.
*   **Type:** pandas DataFrame

### `fumehood_list`

*   **Purpose:** Global variable containing the fumehood list.
*   **Type:** pandas DataFrame
```

Fumehood Sash Height Real-Time Data

Manual Update

Refresh Latest Data

Top Ten Search

Overnight Sash Height
Above Safe Height
1 2 3 4 Good Category Data

Search

Select a fumehood:
FH475 Type your Fumehood
Select a start date: 10/08/2024 Select an end date: 17/08/2024 Search

Frontend interface workflow:

Manual refresh update data, wait for 2-3 minutes and the page would return the status.

It would also return the number of invalid and unavailable fume hoods.

Click on Overnight Sash Height to get the plot of top10 Overnight Sash Height mean value fume hoods.

Click on Above Safe Height to get the plot of top10 count of sash height value over 550 fume hoods.

Click on Category Data to get the plot of such categorization fume hoods base on the model classification from Model.py

Search function would get the plot base on selected fume hood, start and end date.